ParserHandler

* Parses the xml file, stems words, and passes relevant information to the Indexhandler and DocumentHandler.
* Expected inputs are xml files.
* Expected outputs are stemmed words, document ids, author names, title of the page, and the date the xml file was created or last edited.
* ParserHandler(IndexHandler\*, DocumentHandler\*)
  + Constructor for the ParserHandler. Takes in the IndexHandler and DocumentHandler object pointers that it will pass information to.
* void parse(string inputFile);
  + Parses the xml file given. Pases the stemmed words, document ids, author names, title of the pages, and the dates to the designated IndexHandler and DocumentHandler.
  + Uses the Porter2 stemmer from the stemmer.cpp/h file. Removes stopwords.
* getPageCount()
  + Gets the number of pages the particular ParserHandler object has processed

QueryHandler

* Takes in the query from user input handled by the UI and returns a list of the relevant information prompted.
  + QueryHandler(IndexHandler\*)
    - Takes in a pointer to the IndexHandler object and constructs a QueryHandler object
  + list<DocNode\*>\* makeQuery(string);
    - Takes in a string containing the query as input, outputs a list of DocNode\*s that fulfill the query. May contain terms and keywords.

UserInterface

* Acts as a mediator between the user and the components of the search engine, handling all user input and terminal output.
  + UserInterface();
    - Creates the various components used by the search engine, stores pointers to these.
  + ~UserInterface()
    - Cleans up any dynamic memory left over from operation.
  + int startInterface();
    - The bulk of the UI, handling the operation of the search engine
    - Takes input from the user, handles it in context to the current state of the UI
    - Produces output to the terminal depending upon the user’s requests.

IndexIO

* Handles the persistent index, printing it to a file and retrieving for the IndexHandler on command.
  + IndexIO(IndexHandler\*);
    - Sets a pointer to the index to input to and output from
  + void outputIndex();
    - Outputs the index to a file in a specific format.
    - Input comes from the Index
    - Outputs to a file in the working directory
  + void inputIndex();
    - Creates an index from a persistent index file
    - Input comes from a file in the working directory
    - Outputs ListNode\* ‘s to the IndexHandler, to populate the index

DocumentHandler

* Handles the input and output of document data, including the title, author, date, and full text of each parsed document. Uses a map of stream positions to quickly find data from a given id.
  + DocumentHandler(int);
    - Takes an integer representing how many files the documents between
  + ~DocumentHandler();
    - Clears the stream position map
  + string getFullText(int);
    - Gets the Title, Author, Date, and Text from a file determined by the passed in id
    - Takes an document id as input
    - Outputs a string containing the requested data
  + string getQuery(int);
    - Gets the Title, Author, and Date from a file determined by the passed in id
    - Takes an document id as input
    - Outputs a string containing the requested data
  + void addDocument(int,string,string,string,string);
    - Takes the input given and prints it to a file, saving stream positions to retrieve it quickly at a later time.
    - Takes an document id and text fields (title, author, date, text) as input
    - Outputs the information to a file determined by the id
  + void printPositions();
    - Prints the stream positions in the stream position map to a file for persistent use
  + void readPositions();
    - Reads the stream positions in the stream position map from a file for persistent use
  + void clearIndex();
    - Clears the files used by the class.

IndexHandler

* Handles the basic input and retrieval of the index
  + IndexHandler(int);
    - Sets the kind of data structure to use (0 for Avl Tree, 1 for Hash Table)
  + ~IndexHandler();
    - Clears the index
  + void addDocId();
    - Sets the current document id to insert (to minimize id passes)
  + void insert(string&)
    - Inserts a string into the index (from parser)
  + void insert(ListNode\*)
    - Inserts a ListNode\* into the index (from persistent index)
  + list<DocNode\*>\* search(string&)
    - Locates and returns a list of DocNode\* ‘s that contain the inputted word
  + int getInsertCount()
    - Returns the number of total inserts made into the index
  + int getUniqueCount()
    - Returns the total number of unique word inserts into the index
  + string printNodes()
    - Creates a string in a special format from output to the persistent index
  + string printTopWords()
    - Compiles a list of the top fifty words in the index, returns a string containing them
  + void setPageCount(int);
    - Sets the page count (for calculating the Idf value)
  + int getPageCount();
    - Returns the page count currently stored in the index.

IndexInterface

* An abstract class used as the basis for both the AvlTree and the Hash Table. The methods listed below will be explained in depth in the structure documentation.
  + IndexInterface()
  + virtual void insert(string&, int&) = 0;
  + virtual void insert(ListNode\*) = 0;
  + virtual ListNode\* search(string&) = 0;
  + virtual int getUniqueCount() = 0;
  + virtual string printNodes() = 0;
  + virtual void compileNodes(vector<ListNode\*>&) = 0;

HashTable

* A basic Hash Table implementation with Avl Trees in its buckets.
  + HashTable(int);
    - Builds the AvlTree table with the given number of buckets.
  + ~HashTable();
    - Deletes the table buckets
  + void insert(string&,int&)
    - inserts the inputted string and int into the index, using a hash function to calculate the bucket to place the element in.
  + void insert(ListNode\*)
    - inserts the inputted ListNode\* into the index, using a hash function to calculate the bucket to place the element in.
  + ListNode\* search(string&)
    - returns a ListNode\* corresponding to the inputted, which comes from the same bucket that the hash function would place the string in.
  + int getUniqueCount();
    - returns the number of unique words inserted into the index, calculated from the sum of the Avl Tree buckets.
  + string printNodes();
    - Compiles a string containing pertinent data for the persistent index, built from the Avl Tree buckets.
  + void compileNodes(vector<ListNode\*>&);
    - Compils a vector of the ListNode\* ‘s contained in the index.

AVLTree

* A basic Avl Tree implementation, designed to be self balancing.
  + AVLTree()
    - sets the root node to null, prepares the tree for input
  + ~AVLTree()
    - delete the nodes in the tree recursively
  + void insert(string&,int&);
    - recursively finds and inserts the new element into the tree, balancing as needed.
  + void insert(ListNode\*);
    - recursively finds and inserts the new element into the tree, balancing as needed.
  + ListNode\* search(string&);
    - recursively finds and returns element matching to the inputted string.
  + int getUniquecount();
    - retrieves the number of unique words inserted into the Avl tree
  + string printNodes();
    - compiles the nodes recursively to print to the persistent index
  + void compileNodes(vector<ListNode\*>&);
    - Compiles a vector of the ListNode\* ‘s in the index.

ListNode

* Holds a word, a list of document ids corresponding to the word, and the total count that the word has appeared.
  + ListNode(string&,int&);
    - Builds the ListNode using the string as the word and the int as the first node.
  + ListNode(string&,int&, int&, int&);
    - Builds the ListNode using the string as the word, first int as total count, second int as the first id, and third int as the count for that id.
  + ~ListNode();
    - Deletes the DocNode list and its contents.
  + list<DocNode\*>\* getList();
    - retrieves the list of DocNode\* ‘s.
  + string getWord();
    - Returns the word the ListNode corresponds to.
  + int getCount();
    - Returns the total count of the word
  + void insertElement(int&);
    - Insert a new document id, incrementing its count if it has appeared before
  + void insertElement(int&,int&);
    - Insert a new document id and setting its count to the second value.
  + string printList();
    - Creates a string containing the word, count, all ids and associated count for the persistent index.

DocNode

* Holds an id, a count for that id, and a TfIdf value for that id. Acts as the smallest unit of the data structure.
  + DocNode(int&)
    - Creates a new node with an id and an initial count of 1.
  + DocNode(int&,int&)
    - Create a new node with an id and a initial count equal to the second argument.
  + int getID();
    - Return the id of the DocNode
  + int getCount();
    - Return the count of the DocNode
  + double getTfIdf();
    - Return the Tf-Idf value of the DocNode
  + void increaseCount();
    - Increment the value of count by one.
  + void setTfIdf(double idf);
    - Calculate the Tf-Idf value given the corresponding word’s idf value.
  + void mergeNode (DocNode\*);
    - Combine the Tf-Idf value of a given node pointer with the Tf-Idf value of this node. For querying purposes.